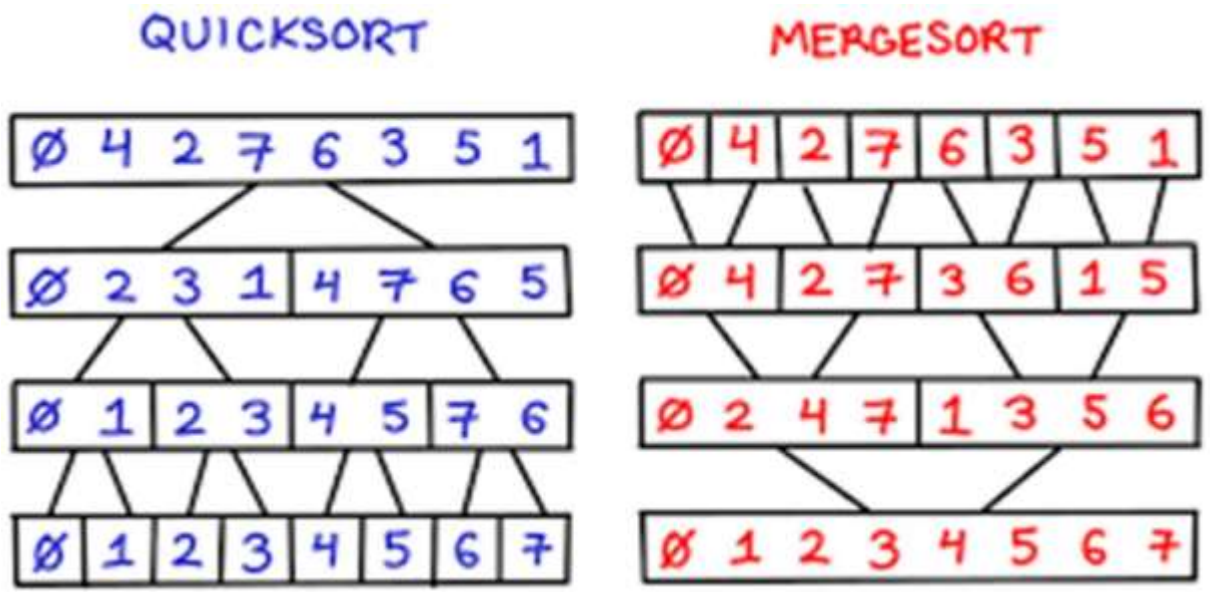
SNS COLLEGE OF TECHNOLOGY

(Autonomous)

COIMBATORE-35

Quick sort,Merge sort



Quick Sort

- Quick sort follows **Divide and Conquer** algorithm. It is dividing elements in to smaller parts based on some condition and performing the sort operations on those divided smaller parts. Hence, it works well for large datasets. So, here are the steps how Quick sort works in simple words.
- First select an element which is to be called as **pivot** element.
- Next, compare all array elements with the selected pivot element and arrange them in such a way that, elements less than the pivot element are to it's left and greater than pivot is to it's right.
- Finally, perform the same operations on left and right side elements to the pivot element.
- So, that is the basic outline of Quick sort. Here are the steps which need to be followed one by one to perform Quick sort.

Quick sort

How does QuickSort Work

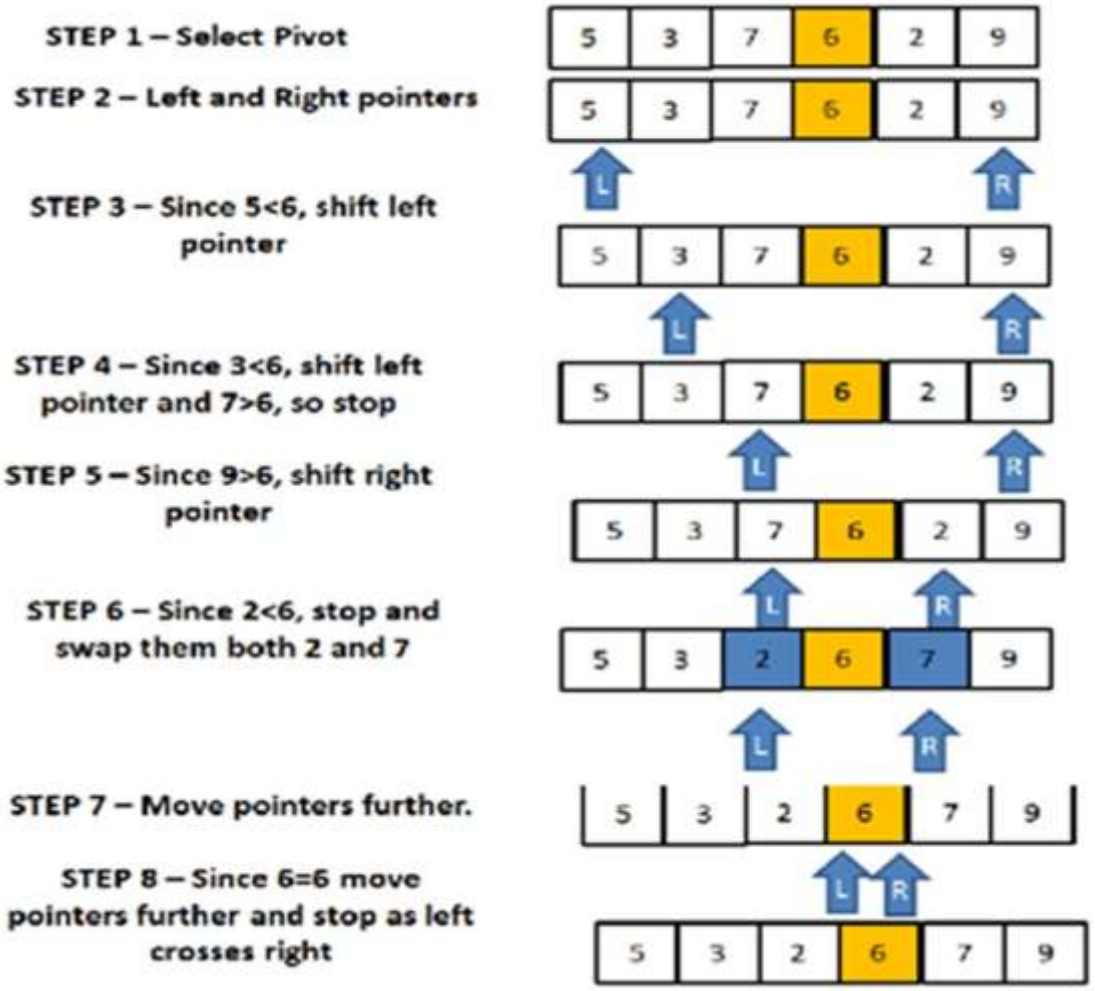
- First find the "**pivot**" element in the array.
- Start the left pointer at first element of the array.
- Start the right pointer at last element of the array.
- Compare the element pointing with left pointer and if it is less than the pivot element, then move the left pointer to the right (add 1 to the left index). Continue this until left side element is greater than or equal to the pivot element.
- Compare the element pointing with right pointer and if it is greater than the pivot element, then move the right pointer to the left (subtract 1 to the right index). Continue this until right side element is less than or equal to the pivot element.

Quick sort

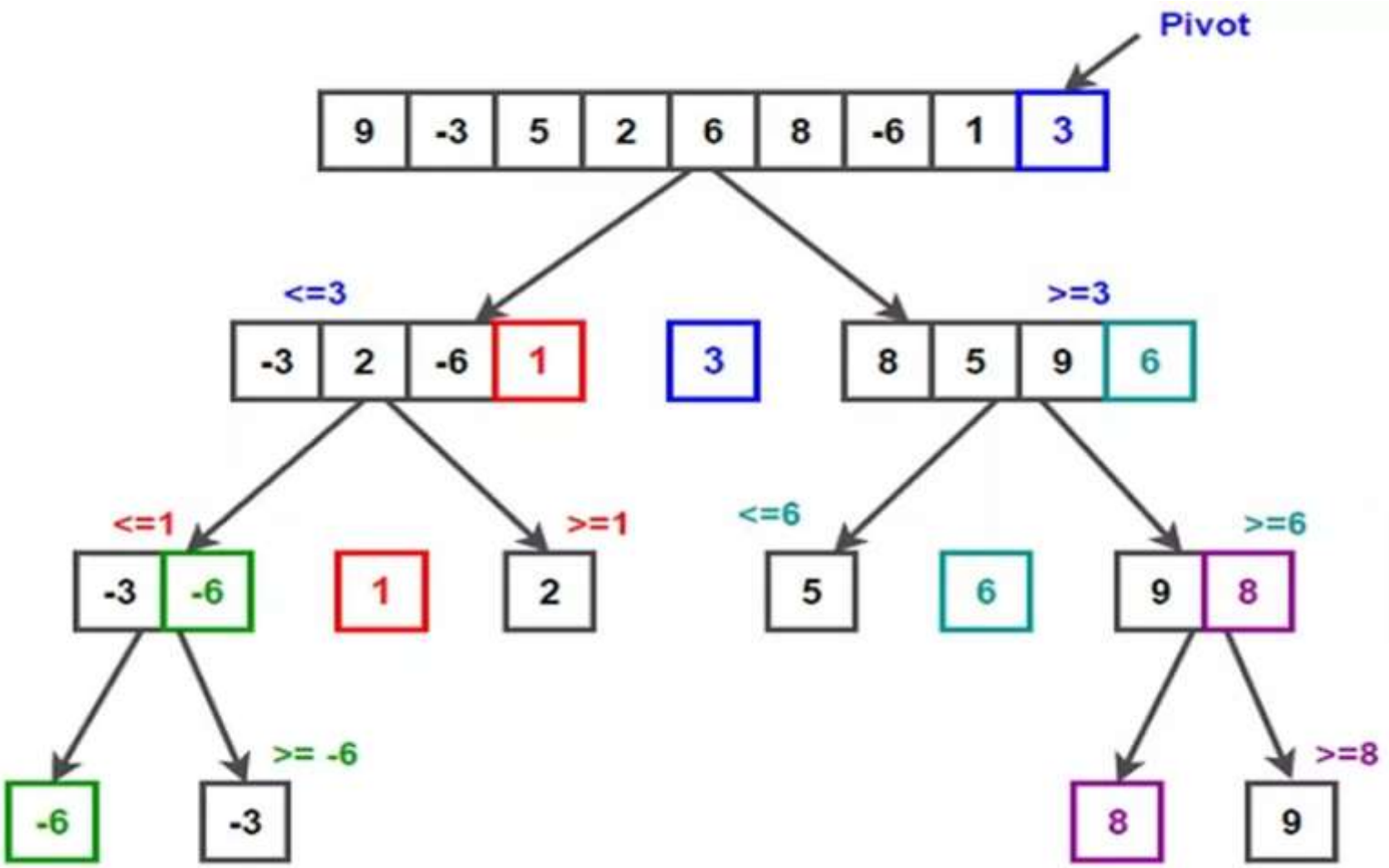
How does QuickSort Work

- Check if left pointer is less than or equal to right pointer, then saw the elements in locations of these pointers.
- Increment the left pointer and decrement the right pointer.
- If index of left pointer is still less than the index of the right pointer, then repeat the process; else return the index of the left pointer.

Quick sort


STEP 1 – Select Pivot
STEP 2 – Left and Right pointers
STEP 3 – Since 5<6, shift left pointer
STEP 4 – Since 3<6, shift left pointer and 7>6, so stop
STEP 5 – Since 9>6, shift right pointer
STEP 6 – Since 2<6, stop and swap them both 2 and 7
STEP 7 – Move pointers further.
STEP 8 – Since 6=6 move pointers further and stop as left crosses right
5 3 7 6 2 9
L
R

Quick sort



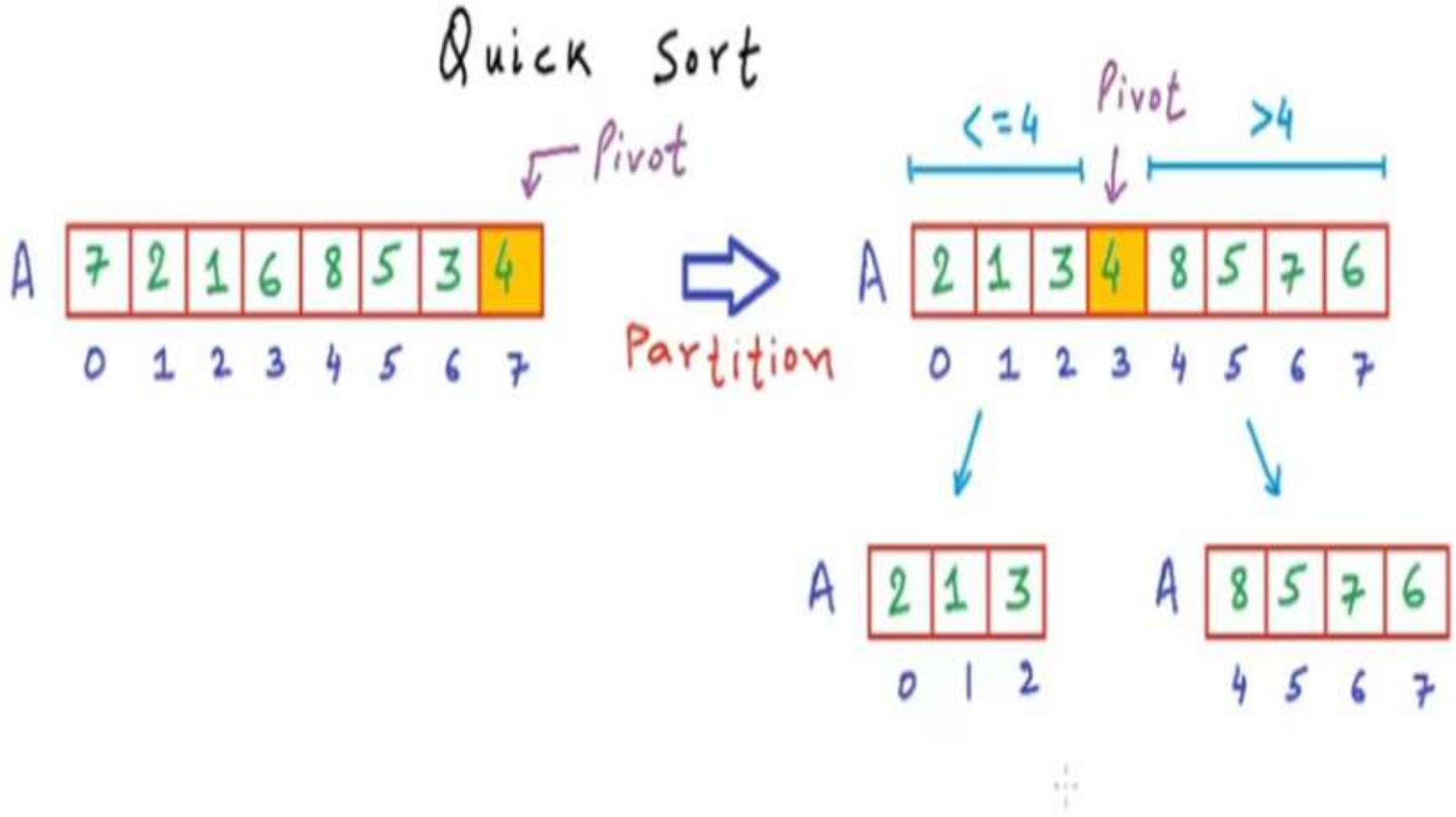
Quick sort

```
function partition(items, left, right) {
    var pivot   = items[Math.floor((right + left) / 2)], //middle element
        i       = left, //left pointer
        j       = right; //right pointer
    while (i <= j) {
        while (items[i] < pivot) {
            i++;
        }
        while (items[j] > pivot) {
            j--;
        }
        if (i <= j) {
            swap(items, i, j); //swap two elements
            i++;
            j--;
        }
    }
    return i;
}
```

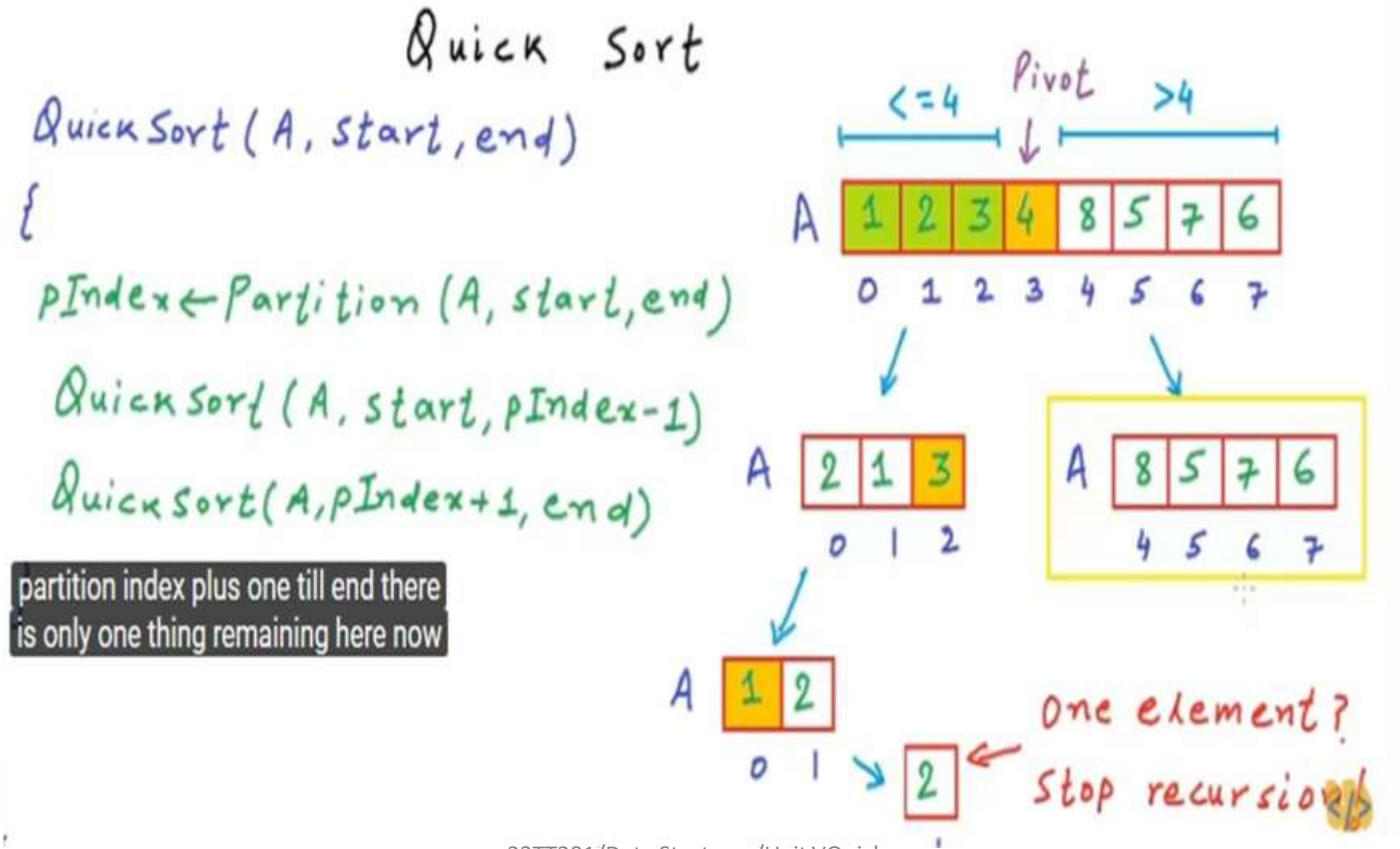
Quick sort

```
function quickSort(items, left, right) {
    var index;
    if (items.length > 1) {
        index = partition(items, left, right); //index returned from partition
        if (left < index - 1) { //more elements on the left side of the pivot
            quickSort(items, left, index - 1);
        }
        if (index < right) { //more elements on the right side of the pivot
            quickSort(items, index, right);
        }
    }
    return items;
}
// first call to quick sort
var result = quickSort(items, 0, items.length - 1);
```

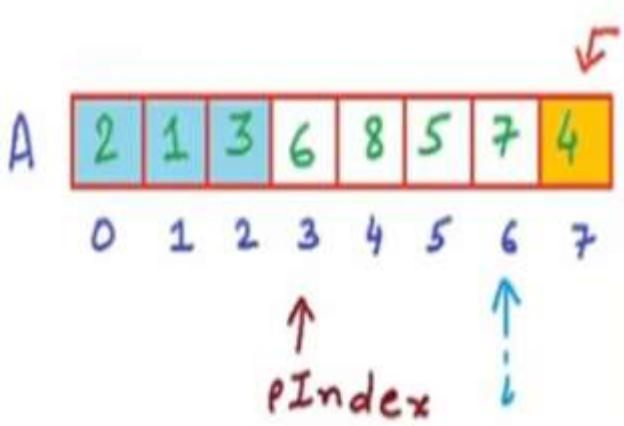
Quick sort Example



Quick sort Example



Quick sort Example




```
Partition (A, start, end)
{
  pivot ← A[end]
  pIndex ← start
  for i ← start to end-1
  {
    if (A[i] <= pivot)
    {
      swap (A[i], A[pIndex])
      pIndex ← pIndex + 1
    }
  }
```

Quick sort Example

```
int Partition(int *A,int start,int end) {
    int pivot = A[end];
    int partitionIndex = start; // set partition index as start initially
    for(int i = start;i<end;i++) {
        if(A[i]<= pivot) {
            swap(A[i],A[partitionIndex]); // swap if element is lesser than pivot
            partitionIndex++;
        }
    }
    swap(A[partitionIndex],A[end]); // swap pivot with element at partition index
    return partitionIndex;
}
void QuickSort(int *A,int start,int end) {
    if(start < end) {
        int partitionIndex = Partition(A,start,end); // calling partition
        QuickSort(A,start,partitionIndex-1);
        QuickSort(A,partitionIndex+1,end);
    }
}
int main() {
    int A[] = {7,2,1,6,8,5,3,4};
    QuickSort(A,0,7);
    for(int i =0;i<8;i++) cout<<A[i]<<" ";
}
```

and there is no way you can perform the merge process without using auxiliary arrays

Merge sort

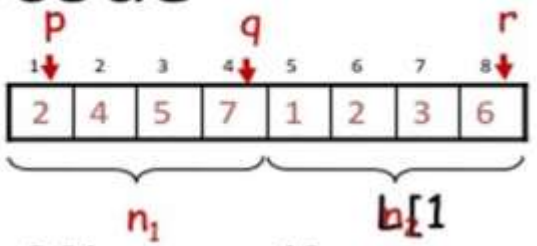
- The merge sort algorithm is a divide and conquers algorithm.
- In the divide and conquer paradigm, a problem is broken into smaller problems where each small problem still retains all the properties of the larger problem -- except its size. To solve the original problem, each piece is solved individually; then the pieces are merged back together

Merge sort

Merge - Pseudocode

Alg.: MERGE(A, p, q, r)

1. Compute n_1 and n_2
2. Copy the first n_1 elements into
   $L[1\ .\ .\ n_1 + 1]$ and the next n_2 elements into $R[1\ .\ .\ n_2 + 1]$
3. $L[n_1 + 1] \leftarrow \infty$; $R[n_2 + 1] \leftarrow \infty$
4. $i \leftarrow 1$; $j \leftarrow 1$
5. **for** $k \leftarrow p$ **to** r
6. **do if** $L[\,i\,] \leq R[\,j\,]$
7. **then** $A[k] \leftarrow L[\,i\,]$
8. $i \leftarrow i + 1$
9. **else** $A[k] \leftarrow R[\,j\,]$
10. $j \leftarrow j + 1$



11

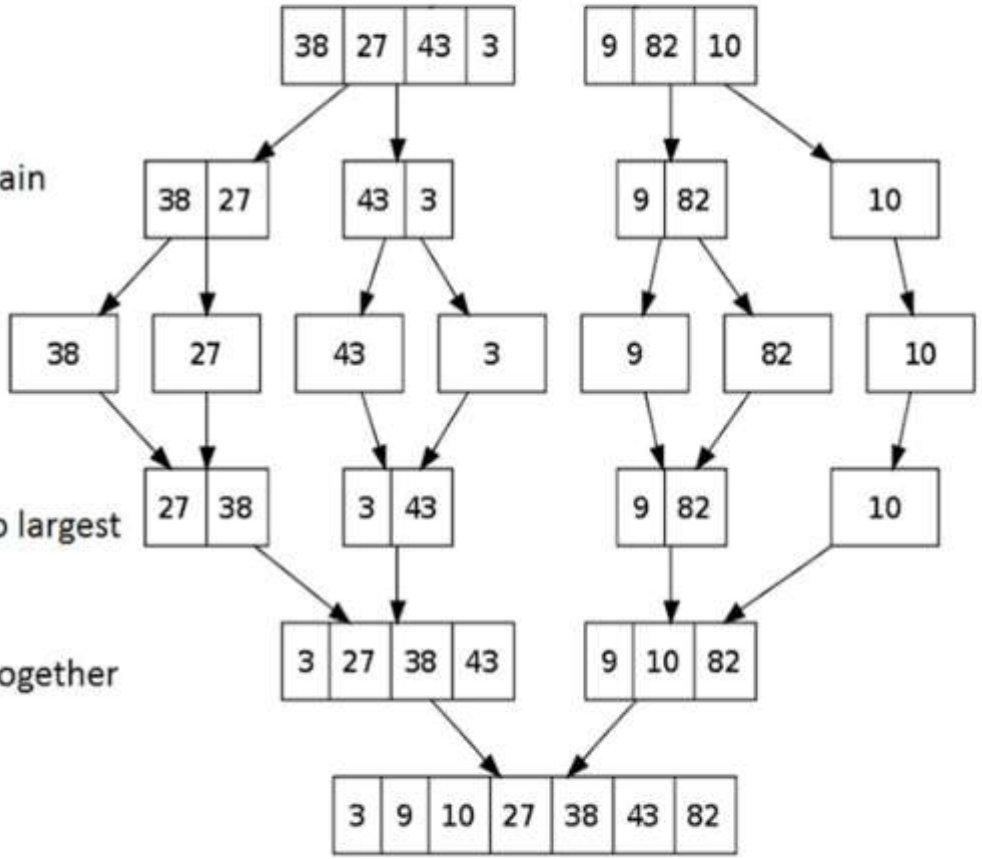
Merge sort

```
IF low < high
   THEN mid = (low + high)/2
        MERGE_SORT (A, low, mid)
        MERGE _SORT(A, mid + 1, high)
        MERGE (A, low, mid, high)

MERGE (A, low, mid, high )

   n1 ← mid − low + 1
   n2 ← high − mid
   Create arrays L[1 . . n1 + 1] and R[1 . . n2 + 1]
   FOR i ← 1 TO n1
       DO L[i] ← A[low + i − 1]
   FOR j ← 1 TO n2
```

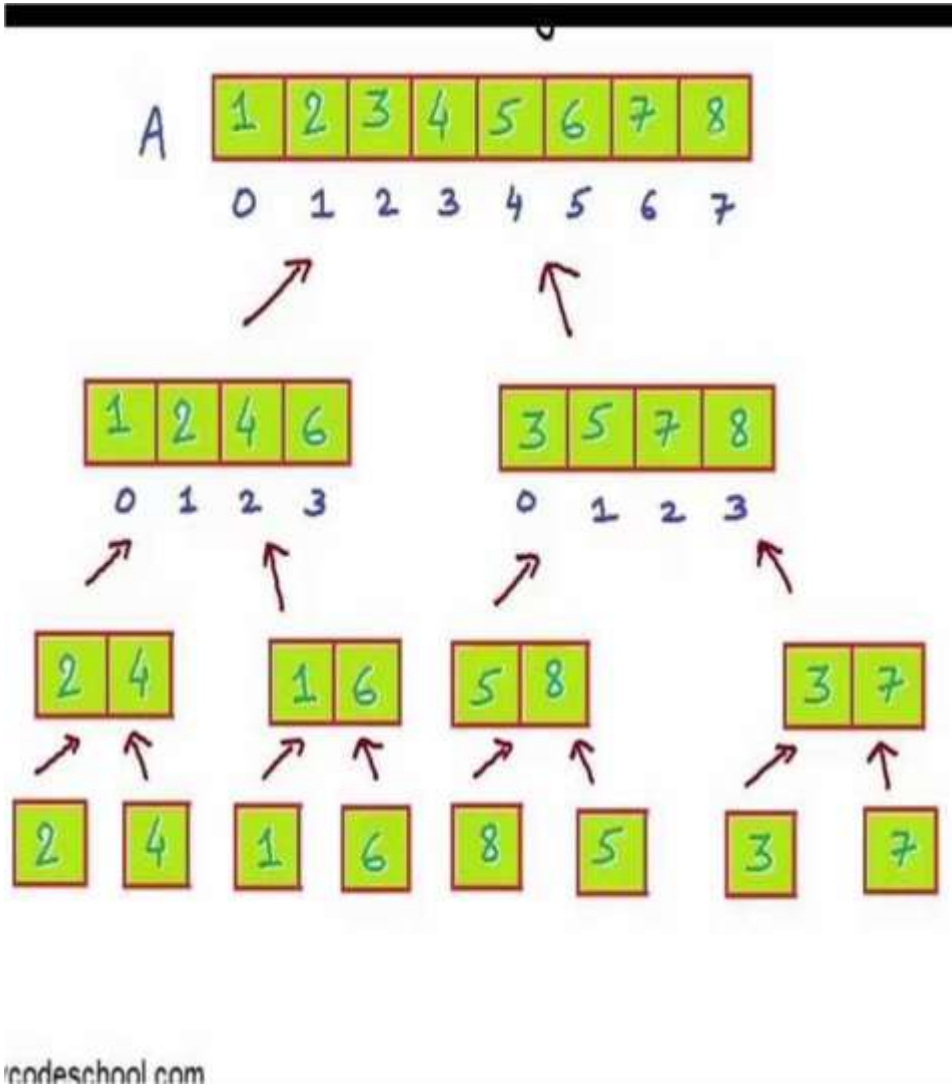
Merge Sort

How MergeSort Algorithm Works Internally

1. Divide the array into two parts
2. Divide the array into two parts again
3. Break each element into single parts
4. Sort the elements from smallest to largest
5. Merge the divided sorted arrays together
6. The array has been sorted



Merge sort Example




```
Mergesort (A)
{ n ← length(A)
  if (n < 2) return
  mid ← n/2
  left ← array of size(mid)
  right ← array of size(n-mid)
  for i ← 0 to mid-1
      left[i] ← A[i]
  for i ← mid to n-1
      right[i-mid] ← A[i]
  Mergesort (left)
  Mergesort (right)
  Merge (left, right, A)
```

Time Complexity Comparision

Algorithms	Average	Worst	Space
Bubble	$O(n^2)$	$O(n^2)$	$O(1)$
Select	$O(n^2)$	$O(n^2)$	$O(1)$
Insert	$O(n^2)$	$O(n^2)$	$O(1)$
Merge	$O(n \log n)$	$O(n \log n)$	$O(n)$
Quick	$O(n \log n)$	$O(n^2)$	$O(\log n)$